Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

When do agents outperform centralized algorithms?

A systematic empirical evaluation in logistics

Rinde R.S. van Lon · Tom Holvoet

Received: July 15, 2016 / Accepted: June 14, 2017*

Abstract Multi-agent systems (MAS) literature often assumes decentralized MAS to be especially suited for dynamic and large scale problems. In operational research, however, the prevailing paradigm is the use of centralized algorithms. Present paper empirically evaluates whether a multi-agent system can outperform a centralized algorithm in dynamic and large scale logistics problems. This evaluation is novel in three aspects: 1) to ensure fairness both implementations are subject to the same constraints with respect to hardware resources and software limitations, 2) the implementations are systematically evaluated with varying problem properties, and 3) all code is open source, facilitating reproduction and extension of the experiments. Existing work lacks a systematic evaluation of centralized versus decentralized paradigms due to the absence of a real-time logistics simulator with support for both paradigms and a dataset of problem instances with varying properties. We extended an existing logistics simulator to be able to perform real-time experiments and we use a recent dataset of dynamic pickup-and-delivery problem with time windows instances with varying levels of dynamism, urgency, and scale. The OptaPlanner constraint satisfaction solver is used in a centralized way to compute a global schedule and used as part of a decentralized MAS based on the dynamic contract-net protocol (DynCNET) algorithm. The experiments show that the DynCNET MAS finds solutions with a relatively lower operating cost when a problem has all following three properties: medium to high dynamism, high urgency, and medium to large scale. In these circumstances, the centralized algorithm finds solutions with an average cost of 112.3% of the solutions found by the MAS. However, averaged over all scenario types, the average cost of the centralized algorithm is 94.2%. The results indicate that the MAS performs best on very urgent problems that are medium to large scale.

* The final publication is available at Springer via doi:10.1007/s10458-017-9371-y

Rinde R.S. van Lon · Tom Holvoet
imec-DistriNet, KU Leuven
3001 Leuven, Belgium
E-mail: Rinde.vanLon@cs.kuleuven.be
E-mail: Tom.Holvoet@cs.kuleuven.be

Keywords Multi-agent systems · Agents · Centralized · Decentralized · Empirical · Evaluation · Dynamism · Urgency · Scale · Operational research · Logistics

1 Introduction

Multi-agent systems [1, 2] is a broad research area involving autonomous software entities, called agents, that typically have a local view of the world. Areas include decentralized control systems, agent based simulation, game theory, trust & reputation, negotiation, etc. In the present paper we use multi-agent systems (MASs) as a paradigm for designing decentrally controlled systems. MASs have been applied in numerous industrial deployments as described by Pěchouček and Mařík [3]. One category of deployments involves operational research (OR) and logistics. For example, Weyns et al. [4] describe an application of MAS technology for operating automated guided vehicles in a warehouse and Dorer and Calisti [5] describe a MAS for dynamic transport optimization.

The focus in the present paper is on dynamic logistics, more specifically, on dynamic pickup and delivery problems (PDPs) [6]. Although literature reports on various studies on applying MASs for dynamic PDP, no systematic evaluation has been conducted that allows to draw conclusions of benefits or limitations of MAS approaches compared to centralized approaches. The aim of the present paper is to systematically evaluate both approaches with varying levels of dynamism, urgency, and scale.

1.1 Multi-agent systems related work

Fischer et al. [7] were one of the first to compare a decentrally controlled MAS with centralized OR heuristics in logistics. In their paper, the authors use a natural mapping of agents to the problem domain, a truck agent is responsible for a single vehicle and a shipping company agent is responsible for handing out new tasks. These agents participate in a dynamic version of contract-net protocol (CNET) first introduced by Smith [8]. Fischer et al. [7] report that the centralized and the decentralized approaches have similar performance but the decentralized approach performs relatively better when the tasks are more urgent. The authors speculate that this might be a general property of contract-net-like algorithms, but they recognize that this speculation must be confirmed by more empirical experiments, such as the one presented in present paper.

In a similar spirit, Mes et al. [9] evaluated an agent-based scheduling approach and look-ahead heuristics for a real-time transportation problem on an underground transport network. In their study, the authors varied several problem properties such as time between orders (related to degree of dynamism), time window length (related to urgency) and the number of nodes in the network (related to scale). The look-ahead heuristics that they used are LocalControl and SerialScheduling. Unfortunately Mes et al. do not specify the exact definitions of the heuristics, hindering the reproducibility of their work. The experimental results show that the agent-based approach always outperforms the look-ahead heuristics. These results are very interesting, especially when considering that MASs are not used as often in logistics compared to centralized algorithms.

In 2008, Máhr et al. [10] did a similar comparison but used a mixed integer program (MIP) instead of simple heuristics. The authors used an auction based coordination mechanism similar to CNET. Their results show that the MAS based approach and the MIP based approach perform comparable in dynamic problem instances. However, compared to the present paper, the problem size used by Máhr et al. is relatively small. The dataset of van Lon and Holvoet [11] that we use contains instances that are 2 to 18 times larger. Interestingly, Máhr et al. suggest, for future work, to do a similar experiment but on differing problem sizes, which is, among other things, what we do in the present paper.

In a subsequent work from 2010, Máhr et al. [12] focused on two types of uncertainty in their problem definition: service time uncertainty and job arrival uncertainty. The results obtained by the authors were mixed. With high service time uncertainty the agent based approach performs better, while in the case of extreme service time combined with job arrival uncertainty the centralized optimization approach outperforms the agent-based approach. However, in the setup by Máhr et al. the urgency of the tasks is variable, it is unclear how this variation influences the result. Therefore, their experiment provides limited insight in the influence of specific problem properties on the effectiveness of centralized and decentralized approaches. This is contrary to the experiments described in the present paper where we systematically investigate the different problem properties explicitly.

The works described above have several shortcomings that hinder the advancement of the fields of MASs and OR. Firstly, there is no common platform on which centralized and decentralized algorithms can be tested on logistics problems in real-time with a fair allocation of hardware resources. Such a simulation platform would facilitate evaluations of algorithms from both the MASs and OR domains, allowing researchers to focus on the improvement of the algorithms while also learning their relative strengths and weaknesses. Secondly, the previously mentioned work did not publish the datasets, algorithms, and supporting code that was used to conduct experiments. It has been argued before by Ince et al. [13] and van Lon and Holvoet [14] that this is a problem that needs to be addressed as it would aid reproducibility and extensibility of existing research. Ideally, the opening of source code, data, and related tools should be the default state of practice as this increases the accountability and thus the value of this field of scientific research. Thirdly, to be able to investigate the circumstances for which specific algorithms perform better than another, it is paramount to be able to independently vary specific problem properties. Therefore, exact definitions of the problem properties are required, allowing precise measurements of the properties. These measures can then be used to meticulously create problem instances that vary only in the selected problem property. Unfortunately, the previously cited works did not isolate the relevant properties (for example urgency in [12]), this limits the usefulness of the experimental results with respect to properties in the problem.

1.2 Operational research related work

Most of the papers discussed above target a variant of the dynamic PDP. Berbeglia et al. [6] gave an overview of variants of dynamic PDPs. In this paper we target the dynamic pickup and delivery problem with time windows (PDPTW) which is a special case of the dynamic vehicle routing problem (VRP). In these problems,

dynamism is often caused by the arrival of new tasks [15]. At the beginning of a work day, typically only a proportion of tasks is known. In the present paper we consider a *purely* dynamic PDP, no information about tasks is known beforehand. Therefore it is not possible to plan ahead, all computations have to be done online. In general, there are three different centralized approaches to the PDP: exact methods, heuristics, and stochastic modeling or sampling. Exact methods are known to be less scalable than non-exact methods [15]. And, because of the NP-hard nature of PDP, exact methods quickly become infeasible to use. Stochastic modeling or sampling assumes that some a priori information about the future is known, in the present paper we do not assume to have such information. Therefore we focus our description of centralized approaches on heuristics. Heuristics are capable of quickly finding (sub-optimal) solutions. Gendreau et al. [16] developed a dynamic version of tabu search with a neighboring structure based on ejection chains. The algorithm runs in between dynamic changes of the problem and when a vehicle has finished a pickup or delivery. Madsen et al. [17] created an insertion heuristic for a dynamic dial-a-ride-problem (DARP). Several rolling horizon heuristics were investigated by Yang et al. [18], with a rolling horizon, only tasks in the near future, within the time horizon, are considered.

1.3 Objectives

The goal of the current paper is to systematically evaluate the performance of a centralized and a decentralized algorithm in a real-time logistics problem. The algorithms guide a cooperative fleet of vehicles to service dynamically appearing customers while minimizing customer waiting times and vehicle travel times. The aim is not to find the best conceivable algorithm but to get insight into the strengths and weaknesses of equivalent centralized and decentralized algorithms under varying circumstances while constrained by the same amount of computational power. We consider a centralized algorithm equivalent to its decentralized counterpart if they use the same underlying solver of problem instances. The method of control, i.e. centralized or decentralized, determines how the solver is used which is the distinguishing difference between the algorithms. Since the two algorithms are constrained by the same amount of computational power, any performance difference measured between the algorithms can be attributed to their method of control. There are several hypotheses related to the domain of logistics that are of interest for the current paper:

1. A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more dynamic problem instances
2. A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more urgent problem instances
3. A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on larger scale problem instances

Operating cost is defined as a combination of customer waiting times and vehicle travel times. To investigate these hypotheses systematically, it is imperative to formally define the concepts of dynamism, urgency, and scale. Dynamism and urgency have recently been defined by van Lon et al. [19] in the context of dynamic logistics. In short, dynamism is defined as the continuity of change and urgency is defined

as the amount of time that is available to respond to an incoming request. These properties are, together with scale, used by van Lon and Holvoet [11] to define a dataset with varying levels of dynamism, urgency, and scale. The open source logistics simulator RinSim [20] has support for this dataset, allowing easy comparison of centralized and decentralized algorithms. Present paper describes how we use this dataset and simulator to investigate the aforementioned hypotheses.

1.4 Contributions and overview

The formal problem definition and definitions of dynamism, urgency, and scale as defined by the dataset [11] are presented in Section 2. The present paper contributes the following:

- the RinSim simulator is extended such that centralized and decentralized approaches can be compared in a fair manner, each approach receives the same amount of processing power and is subject to the same real-time constraints (Section 3);
- an online centralized optimization algorithm and a decentralized dynamic contract-net protocol (DynCNET) that uses the same problem solver, based on the well known OptaPlanner library, are implemented (Section 4);
- the centralized and decentralized algorithms are systematically evaluated on differing levels of dynamism, urgency, and scale (Section 5); and,
- the code of the simulator, algorithms, and experiments as well as the datasets and results are made available online to allow complete reproducibility and future extension of the present work.

The paper is concluded in Section 6.

2 Dynamic pickup-and-delivery problems

We adopt the definition of dynamic PDPs from the dataset described by van Lon and Holvoet [11]. In PDPs there is a fleet of vehicles responsible for the pickup-and-delivery of items. Dynamic PDP is an online problem. Customer transportation requests are revealed over time, during the fleet's operating hours. It is further assumed that the fleet of vehicles has no prior knowledge about the total number of requests nor about their locations or time windows. In this section, we provide an overview of the existing work about dynamic PDP and the dataset as it serves as a foundation of the evaluation in present paper.

2.1 Formal definition

In [11] a *scenario*, which describes the unfolding of a dynamic PDP, is defined as a tuple:

$$\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle := \text{scenario},$$

where

$$\begin{aligned} [0, \mathcal{T}) &:= \text{time frame of the scenario}, & \mathcal{T} > 0 \\ \mathcal{E} &:= \text{list of events}, & |\mathcal{E}| \geq 2 \\ \mathcal{V} &:= \text{set of vehicles}, & |\mathcal{V}| \geq 1 \end{aligned}$$

$[0, \mathcal{T})$ is the period in which the fleet of vehicles $\mathcal{V}$ has to respond to customer requests. The events, $\mathcal{E}$, represent customer transportation requests. Since we consider the purely dynamic PDPTW, all events are revealed between time 0 and time $\mathcal{T}$. Each event $e_i \in \mathcal{E}$ is defined by the following variables:

$$\begin{aligned} a_i &:= \text{announce time} \\ p_i &:= [p_i^L, p_i^R) = \text{pickup time window}, \; p_i^L < p_i^R \\ d_i &:= [d_i^L, d_i^R) = \text{delivery time window}, \; d_i^L < d_i^R \\ pst_i &:= \text{pickup service time span} \\ dst_i &:= \text{delivery service time span} \\ ploc_i &:= \text{pickup location} \\ dloc_i &:= \text{delivery location} \\ tt_i &:= \text{travel time from pickup location to delivery location} \end{aligned}$$

Reaction time is defined as:

$$r_i := p_i^R - a_i = \text{reaction time} \tag{1}$$

The time window related variables of a transportation request are visualized in Figure 1.

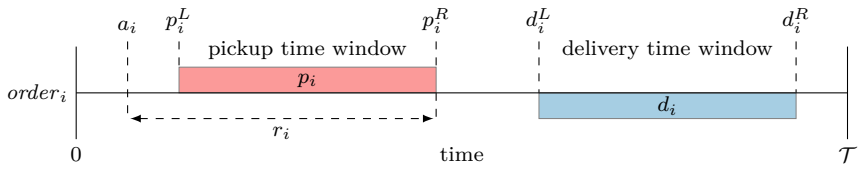


Fig. 1: Visualization of the time related variables of a single order event $e_i \in \mathcal{E}$.

Furthermore it is assumed that:

- vehicles start at a depot and have to return after all orders are handled;
- the fleet of vehicles $\mathcal{V}$ is homogeneous;
- the cargo capacity of vehicles is infinite (e.g. courier service);
- the vehicle is either stationary or driving at a constant speed;
- vehicle diversion is allowed, this means that a vehicle is allowed to divert from its destination at any time;
- vehicle fuel is infinite and driver fatigue is not an issue;
- the scenario is completed when all pickup and deliveries have been made and all vehicles have returned to the depot; and,
- each location can be reached from any other location.

Vehicle schedules are subject to both hard and soft constraints. The opening of time windows is a hard constraint, hence vehicles need to adhere to these:

$$sp_i \geq p_i^L \tag{2}$$

$$sd_i \geq d_i^L \tag{3}$$

sp_i is the start of the pickup operation of order event e_i by a vehicle; similarly, sd_i is the start of the delivery operation of order event e_i by a vehicle. The time window closing (p_i^R and d_i^R) is a soft constraint incorporated into the objective function, it computes the operating cost and needs to be minimized:

$$min := \sum_{j \in \mathcal{V}} \left(vtt_j + td\left\{bd_j, \mathcal{T}\right\}\right) + \sum_{i \in \mathcal{E}} \left(td\left\{sp_i, p_i^R\right\} + td\left\{sd_i, d_i^R\right\}\right) \tag{4}$$

where

$$td\left\{\alpha, \beta\right\} := max\left\{0, \alpha - \beta\right\} \;=\; \text{tardiness} \tag{5}$$

vtt_j is the total travel time of vehicle v_j; bd_j is the time at which vehicle v_j is back at the depot. In summary, the objective function computes the total vehicle travel time, the tardiness of vehicles returning to the depot and the total pickup and delivery tardiness.

2.2 Dataset

Earlier work has argued for, and presented, a dataset characterized by three different properties of dynamic PDPs: dynamism, urgency, and scale [11].

2.2.1 Dynamism

Dynamism is defined in van Lon et al. [19]. Informally, a scenario that changes continuously is said to be dynamic while a scenario that changes occasionally is said to be less dynamic. In the context of PDPTWs a change is an event that introduces additional information to the problem, such as the events in $\mathcal{E}$. Formally, the degree of dynamism, or the continuity of change, is defined as:

$$dynamism := 1 - \frac{\sum_{i=0}^{|\Delta|} \sigma_i}{\sum_{i=0}^{|\Delta|} \bar{\sigma}_i} \tag{6}$$

Δ is the list of event interarrival times:

$$\Delta := \{\delta_0, \delta_1, \ldots, \delta_{|\mathcal{E}|-2}\} = \{a_j - a_i | j = i + 1 \wedge \forall a_i, a_j \in \mathcal{E}\} \tag{7}$$

The interarrival time for a scenario with 100% dynamism is called the perfect interarrival time:

$$\theta := \text{perfect interarrival time} = \frac{\mathcal{T}}{|\mathcal{E}|} \tag{8}$$

Based on this definition, the deviation and maximum possible deviation to the perfect interarrival time can be computed:

$$\sigma_i := \begin{cases} \theta - \delta_i & \text{if } i = 0 \text{ and } \delta_i < \theta \\ \theta - \delta_i + \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\bar{\sigma}_i := \theta + \begin{cases} \dfrac{\theta - \delta_i}{\theta} \times \sigma_{i-1} & \text{if } i > 0 \text{ and } \delta_i < \theta \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

eq. 6 uses the proportion of the actual deviation and the maximum possible deviation. Using this definition the degree of dynamism of any scenario can be computed.

2.2.2 Urgency

In [19] urgency is defined as the maximum reaction time available to the fleet of vehicles in order to respond to an incoming order. Or more formally:

$$\mathit{urgency}\,(e_i) := p_i^R - a_i = r_i \tag{11}$$

To obtain the urgency of an entire scenario the mean and standard deviation of the urgency of all orders can be computed.

2.2.3 Scale

Scale is defined by van Lon and Holvoet [11] as maintaining a fixed objective value per order while scaling the number of orders up in proportion to the number of vehicles in the fleet. Scaling up a scenario $\langle \mathcal{T}, \mathcal{E}, \mathcal{V} \rangle$ with a factor α will create a new scenario $\langle \mathcal{T}, \mathcal{E}', \mathcal{V}' \rangle$ where $|\mathcal{V}'| = |\mathcal{V}| \cdot \alpha$ and $|\mathcal{E}'| = |\mathcal{E}| \cdot \alpha$.

3 A realistic experimentation platform

The dataset presented by van Lon and Holvoet [11] uses the RinSim logistics simulator [20]. In the present paper we intend to quantify the performance of algorithms on scenarios with different properties. Dynamism and urgency are both time related properties, that, in the real world, have a direct impact on the amount of available computation time before an action is required. Scale, on the other hand, is a property that impacts the solution space. Since the dynamic PDP is NP-hard, the problem scale has a significant impact on the amount of time needed for computations. In order to evaluate the impact of these properties on the performance of the algorithms in dynamic PDPs, it is imperative to execute the algorithms in real-time. In a logistics scenario, this means that while vehicles are driving or performing operations the algorithms can compute in parallel. To support a realistic evaluation of the algorithms on the dataset, the RinSim simulator is extended with real-time support. This section first presents an overview of the RinSim architecture followed by the design and evaluation of the real-time extension.

RinSim is a discrete-time logistics simulator that supports running both centralized algorithms and decentralized multi-agent systems. RinSim is written in Java and has a modular design (Figure 2), a `Model` encapsulates a part of a problem domain or algorithm. The simulator can be customized by selecting the models that

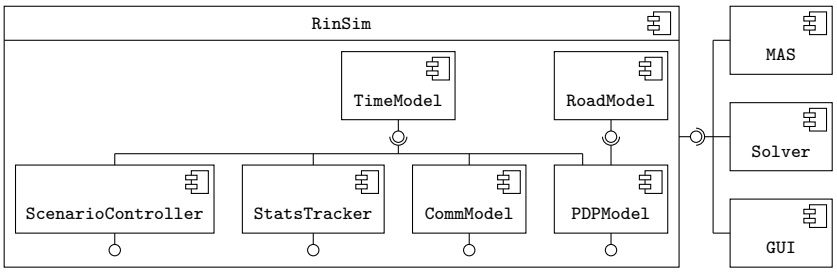


Fig. 2: UML component diagram of RinSim. The simulator subsystem can be configured with a variety of models that all provide some interface. MASs, solvers, and the graphical user interface use these interfaces to interact with RinSim.

are used, this allows simulating a wide variety of logistics problems while maximally reusing existing code. RinSim has a number of built-in models.

`TimeModel` is one of the core models in the simulator, it is responsible for simulating the advancing of time. RinSim is an activity-based simulator, a special case of an event-based simulator [21]. Event-based simulation models a system as it evolves over time, the system state changes at discrete points in time. Events indicate system state changes at specific points in time. Typically, system state changes trigger the scheduling of new events in the future. An advantage of event-based simulation is that time advances are irregular, allowing the simulator to jump over periods with no events. In activity-based simulation, the simulator clock is advanced using a fixed time increment. Any events that are scheduled to occur during this interval are considered to occur at the end of the interval. Therefore, events in activity-based simulations may deviate from their real time. However, the time increment can be chosen to be small enough for these deviations to have no significant effect on the simulations. RinSim is designed with a variety of use cases in mind, one use case is where agents are free to roam around and make ad hoc decisions. When there are many of these agents, the total number of events that need to be scheduled is overwhelming. For that reason, RinSim uses a fixed time increment called a 'tick'. The simulator is initialized with a fixed tick length, for example a tick length of 250 milliseconds. Each tick, RinSim advances the clock and notifies all objects in the simulator that implement the `TickListener` interface (Figure 3). The order in which the `TimeModel` notifies the `TickListener`s is fixed, this ensures that the simulation is deterministic allowing reproducibility of experiments. Each tick, the `TimeModel` hands out a `TimeLapse` instance that indicates the current time and duration of the tick. Each `TickListener` can choose to *consume* the `TimeLapse` and spend it on an action. Once a `TimeLapse` is completely consumed, it can not be used again during that same tick. Using this mechanism RinSim ensures time consistency throughout a simulation.

A `RoadModel` provides an interface for traveling on a road structure. RinSim provides several `RoadModel` implementations, there are graph based implementa-

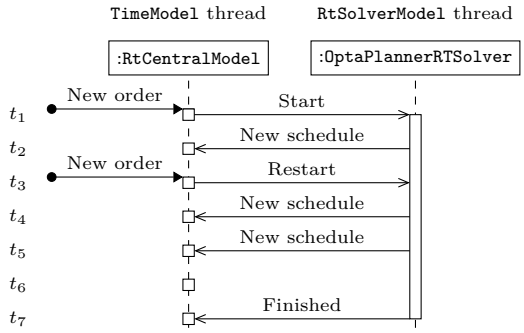


Fig. 3: Execution order of `TickListener`s in the `TimeModel`.

tions that allow objects to traverse a graph. Additionally, there are graph based `RoadModel`s that allow dynamic changes to the graph or that have support for collisions, which allows to simulate a warehouse environment for autonomous guided vehicles. Alternatively, there is a `RoadModel` based on a plane (such as used in [11]) which allows vehicles to travel in a straight line. All `RoadModel`s provide location consistency and ensure that maximum speeds are adhered to.

There are several other major components in RinSim. `PDPModel` is a model that provides simulation of pickup-and-delivery of goods by a vehicle. It ensures capacity and location constraints such that a pickup or delivery operation can only be performed when a vehicle is at the correct location. `CommModel` is a model that provides message based communication to agents, `StatsTracker` records statistics of a simulation run and `ScenarioController` allows the simulation of a predefined scenario (such as the scenarios from [11]).

Figure 2 also shows several external components. The `MAS` component shows how an agent implementation would interact with the simulator. The `Solver` component has a similar interaction with RinSim but both the `MAS` and `Solver` components provide default implementations to aid in the development of the respective algorithms. The `GUI` component provides the RinSim graphical user interface (Figure 4). The RinSim GUI provides several customizable visualizations for different aspects of a simulation.

Besides enforcing consistency inside the simulator models, the code of RinSim itself is meticulously checked by a large number of unit and integration tests (over 1550 tests at the time of writing) to ensure code quality. There are a number of papers reporting on applications of RinSim for scientific experiments. RinSim has been used for simulating bike sharing by Preisler et al. [22, 23], and in our research group for experiments with dynamism and urgency [19], for experiments on the dataset [11], for evolving multi-agent systems for PDPTW [24, 25], and for simulating autonomous guided vehicles in a warehouse [26]. Additionally, RinSim is used at KU Leuven, Belgium, as an educational platform for students in the context of a course on MAS.

3.1 Real-time extension

The standard Java virtual machine (JVM) has no built-in support for real-time execution. However, with a careful software design, the standard JVM can be used

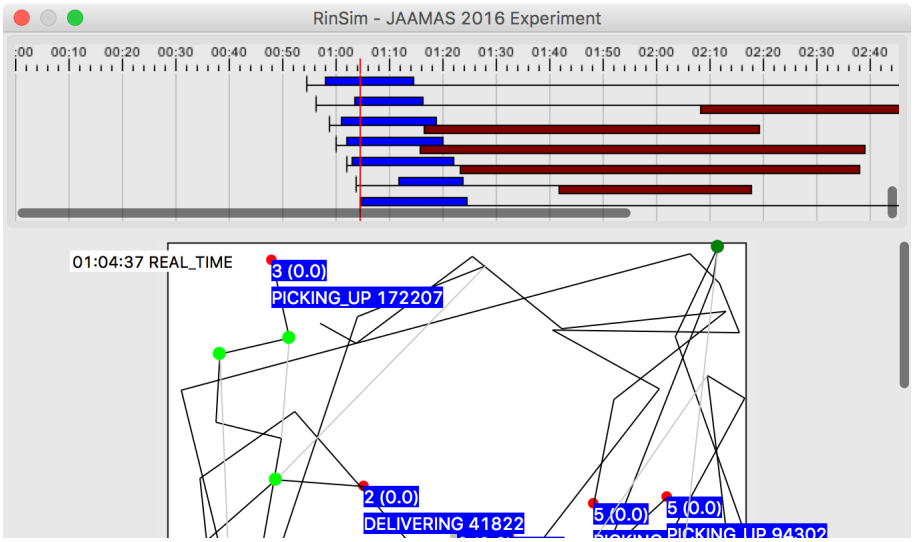


Fig. 4: Screen shot of RinSim. The top part of the screen shows a time window visualization with pickup time windows above the line in blue and delivery time windows below the line in red. The bottom part of the screen shows a two dimensional geographical view of the simulation world. It shows vehicles (red dots), pickup locations (green dots), and the routes vehicles are following (black lines).

to obtain soft real-time behavior. Soft real-time, as opposed to hard real-time, allows occasional deviations from the desired execution timing. In order to obtain acceptable soft real-time behavior, we applied two strategies, first, we minimize the possible deviations from the desired execution timing, and second, we measure and report the actual deviations that occur.

When simulating without real-time constraints, the `TimeModel` will compute all ticks as fast as possible. In a real-time simulator the interval between the *start* of two ticks should be the tick length (e.g. 250 ms). Since the JVM doesn't allow precise control over the timings of threads it is generally impossible to guarantee hard real-time constraints. In real-time mode, RinSim uses a dedicated thread for executing the ticks. If computations need to be done that are expected to last longer than a tick, they must be done in a different thread. RinSim provides a separate model for running solvers in a separate thread called `RtSolverModel`. This minimizes interference of `RtSolverModel` computations with the advancing of time in the simulated world as executed by the `TimeModel`. Additionally, the processor affinity of the threads are set at the operating system level. Setting the processor affinity to a Java thread instructs the operating system to use one processor exclusively for executing that thread. In practice, the actual scheduling of threads on processors depends on the number of available processors and the operating system. Informal tests on a multi core processor running Linux have shown that different threads are indeed run on different processor cores, exactly as specified. By setting the processor affinity of the `TimeModel` thread, deviations from the desired execution timing are minimized.

Nevertheless, time deviations can and do happen because the behavior of the standard JVM can not be controlled completely. In order to be able to measure the possible deviations, RinSim keeps a real-time tick log (Figure 5). In this log the

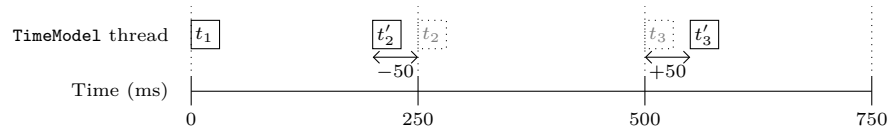


Fig. 5: Illustration of the execution of real-time ticks. In this example tick t'_2 is executed 50 ms earlier than the perfect timing as indicated by tick t_2, tick t'_3 is executed 50 ms later than the perfect timing as indicated by tick t_3.

exact timings (in nanoseconds) of all real-time ticks are kept. With this log, different runs of the simulator can be compared and possible influence on the results can be investigated.

Running a complete logistics simulation in real-time is time consuming, as it will simulate every tick synchronized with real time. However, depending on the specific simulation that is being run, there may be long intervals where no computations are being done other than that of the simulator advancing time in the simulated world. For this reason, RinSim employs a mechanism to dynamically switch between real-time and simulated time (Figure 6). When the simulator is in simulated time, ticks

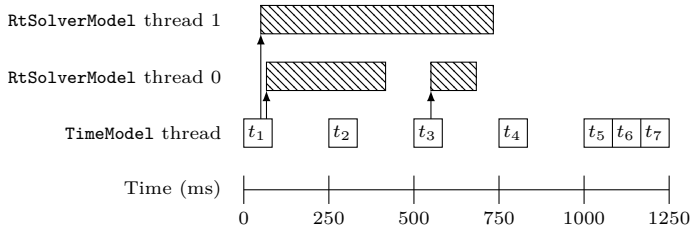


Fig. 6: An example of RinSim with a `RtSolverModel` with two threads. In tick t_1, two solver computation tasks are dispatched in their own threads. In tick t_4 it is detected that all computations have finished, therefore the simulator switches to simulated time in the next tick. Tick t_5, t_6, and t_7 are executed consecutively in simulated time.

will be executed as fast as possible speeding up the simulation significantly. As soon as a computation needs to be done, the simulator must first switch back to real-time mode before this computation can be started.

When a solver starts computing, it receives a snapshot of the current state of the world and starts optimizing the current schedule using that snapshot. Now, the longer a solver is computing, it becomes increasingly likely that the information with which it started computing is outdated. To avoid keeping outdated information for too long, RinSim provides the facility to keep the solver updated in real-time (Figure 7). However, each time the solver thread needs to be updated the solver has to pause for a short period of time, therefore the number of updates should be limited. To balance between the cost of computing based on outdated information and the cost of interrupting the solver thread, the solver is updated only when the problem has changed in a way that changes the search space significantly. The first event which is considered significant is when a new order arrives. A new order must eventually be assigned, so it is important to take this into account as soon as possible. The second significant event is when a vehicle has committed to perform a specific

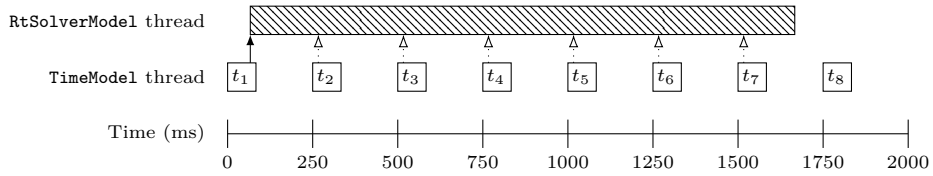


Fig. 7: Graphical depiction of the `TimeModel` updating the solver on every tick. In tick t_1 the solver is started, subsequent ticks can optionally stop, update, and start the solver.

servicing operation. This is important as it fixes a part of the search space, the order that is being serviced can no longer be moved to another vehicle.

3.2 Real-time reliability

Using the log of interarrival times that RinSim keeps, the effect of deviations on the results can be examined. Therefore we did an experiment using three different solvers on the same scenario. We performed 10 repetitions for each algorithm using the same random seed and same scenario. This setup allows to investigate the influence of any deviations of tick interarrival times on the measured scenario cost (using the objective function from Section 2.1). The solvers that were used are a cheapest insertion (CI) heuristic, a first fit decreasing heuristic (FFD), and FFD combined with tabu search (FFD.TABU). Figure 8 shows that FFD.TABU outperforms the simpler heuristics but it introduces some variation in the cost. The tick interarrival

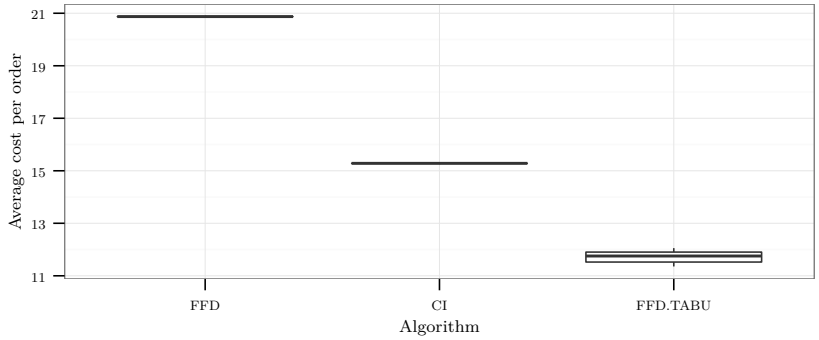


Fig. 8: Boxplots of three OptaPlanner algorithms on a scenario with 10 repetitions with the same random seed, dynamism of 50%, urgency of 20 minutes, and scale 10. FFD.TABU performs best but the cost values are more stochastic.

time logs (Table 1) show that FFD.TABU uses about 10 times more real-time ticks compared to the CI and FFD algorithms. The difference in real-time ticks is expected because FFD.TABU is more complex and therefore requires more computation time. The variation in cost of FFD.TABU is caused by the small variations of tick length that cause small differences in reached solution quality at a specific time in the

Table 1: Accumulated tick interarrival time statistics for the 10 experiments that were conducted for each algorithm. Count indicates the number of real-time ticks, the columns to the right indicate the number of ticks that have a deviation of -10, -1, +1 or +10 milliseconds, respectively, to the expected interarrival time of 250 milliseconds.

Algorithm	μ	σ	Count	-10 ms	-1 ms	$+1$ ms	$+10$ ms
CI	250.0493 ms	2.85 ms	34950	23	168	231	36
FFD	250.0428 ms	3.44 ms	34950	26	193	278	40
FFD.TABU	250.0004 ms	1.73 ms	373055	198	3076	3086	198

simulation. For example, when comparing two simulation runs, a deviation of a single tick may already have an impact on the final result. Consider the situation where due to the deviation of a tick of a few milliseconds a new solution is found by the algorithm one tick later (or earlier), this causes the vehicles to receive the new solution one tick (250 ms) later (or earlier). These small differences may have relatively large effects because the costs that are incurred accumulate over time. Therefore, to minimize the effect of this real-time related stochasticity, a scientific experiment should never rely on a single repetition of a simulation. For that reason, we repeat each simulation three times and we use ten scenarios with the same properties in our experiment setup (Section 5.1).

3.3 Computational fairness

When comparing centralized algorithms with decentralized MASs in a real-time setting, it is important that assignment of computational resources is balanced. For example, both approaches must have the same number of available processor cores, but not necessarily the same number of threads. For this reason, the `RtSolverModel` has a thread grouping option, this binds all solver threads to the same processor core (using processor affinity). When more than one thread is bound to the same processor core, the execution of the threads are interleaved, giving a similar percentage of computation time to each thread. Even though a MAS is typically deployed in a distributed fashion and has therefore access to many processor cores, in the experiments described in this paper the hardware constraints are balanced because the goal is to evaluate centralized and decentralized software paradigms, and not their deployments. In a real-world deployment the hardware constraints of centralized and decentralized approaches most certainly differ from our simulation setup, however, there will invariably be *some* hardware constraints. Therefore, we compare the centralized and decentralized software paradigms irrespective of deployment related hardware constraints.

3.4 Experiments

RinSim has several features to ease running large scale experiments with a lot of individual real-time simulations. RinSim can run multiple simulations in parallel, by giving each simulation its own dedicated set of processor cores, simulations do not affect each other's computational resources. However, this also puts an upper bound

to the number of simulations that can be run in parallel. For example, when twelve cores are available and each simulation requires two cores, the maximum number of simulations that can be run in parallel is five. These five simulations will use ten cores, so that leaves two cores for the operating system to perform background tasks. When an experiment contains more simulations than can be run in parallel, the remainder will be queued by RinSim.

The standard JVM performs just-in-time compilation and adaptive optimization of often used code. These JVM activities can influence the real-time experiments. Therefore, RinSim provides a warm up period that runs several simulations for a predefined time to warm up the JVM. This warm up period reduces the influence on the simulations because the JVM will already be optimized to the code that is going to be run. When running real-time experiments it is recommended to always use a warm up period, as we do (Section 5.1). RinSim also has an option to change the ordering in which individual simulations are run. For example, when two different configurations are tested, it is better to alternate between the configurations instead of first running all simulations with one configuration and then all simulations with another configuration. Alternating the configurations ensures that the individual configurations are subject to similar fluctuations in computation speed and memory availability that are beyond the control of the JVM.

4 Algorithms

To evaluate centralized and decentralized algorithms it is important that the quality of the algorithms is comparable. Comparing a strong centralized algorithm with a weak MAS will not yield meaningful results. For that reason, we use the same solver algorithms framework in both the centralized as well as the decentralized setting. The evaluation focuses on how the algorithms are used, not on the specific algorithms that are used. For the centralized algorithm we have chosen the well known OptaPlanner framework created by De Smet et al. [27]. For the decentralized algorithm we have chosen a multi-agent system with DynCNET because it is a proven technology that has been applied numerous times for task allocation optimization in the context of manufacturing [28]. DynCNET uses the same solver from the OptaPlanner framework, but in a decentralized fashion.

4.1 Centralized algorithm

OptaPlanner [27] is an open source Java constraint satisfaction engine that optimizes planning problems. The project is developed by De Smet et al. [27] and sponsored by RedHat. OptaPlanner provides a wide range of optimization algorithms such as construction heuristics and metaheuristics. It has support for many problem domains such as scheduling and vehicle routing.

OptaPlanner allows customization of the problem definition to that of the PDPTW (Section 2) as is used in the dataset. OptaPlanner is incorporated into RinSim using the `RtCentralModel` and `OptaPlannerRtSolver` classes, the model controls all vehicles centrally using the schedule computed by the solver (Figure 9). The `OptaPlannerRtSolver` continuously updates the `RtCentralModel` of its progress and the model restarts the solver when the problem definition changes (i.e. when a

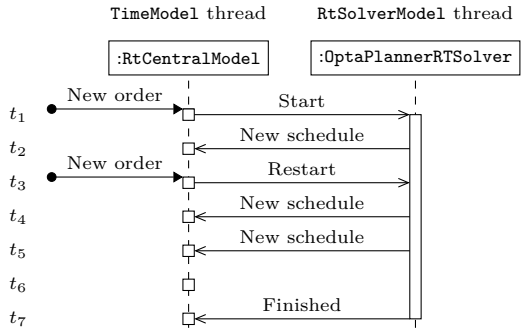


Fig. 9: UML interaction diagram of communication between `RtCentralModel` and `OptaPlannerRtSolver`. In this example the solver is interrupted at t_3, the problem definition is updated with the new order after which the solver continues its search. In t_4 and t_5 it has found a new improving schedule. In t_7 the solver is done computing as it couldn't find an improving schedule anymore.

new order is announced). The continuous updates of the solver use the RinSim mechanism as described in Section 3. This setup allows real-time control of the fleet of vehicles and avoids unnecessary long computations based on outdated information.

OptaPlanner requires a stop condition to halt its search process. We are using the *unimproved time spent* stop condition, that halts the search when the best score has not improved for an amount of time.

4.2 Decentralized multi-agent system

The multi-agent system that is investigated is an implementation of the DynCNET presented by Weyns et al. [29]. DynCNET is a dynamic extension of the CNET first proposed by Smith [8]. Inspired by how companies use subcontracting to collaboratively solve problems, CNET uses contracting to approach the task assignment problem. In CNET, the agent that tenders a task is called the *manager* and it sends a task announcement to potential *contractors*. Each potential contractor can either ignore the announcement or send a *bid* to the manager. The manager then selects its best bid and *awards* the task to the contractor. Figure 10 shows the UML interaction diagram for the CNET auction process. Although an auction can be, and usually

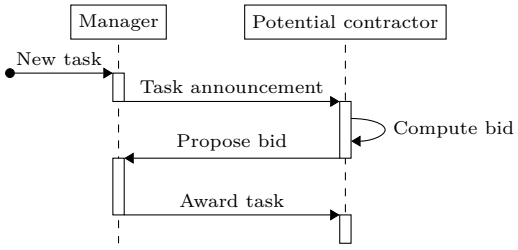


Fig. 10: UML interaction diagram of a CNET auction.

is, used in a competitive setting, we use auctions in a purely cooperative setting. We assume that both the contractors and the manager are working for the same company. The dynamic extension of CNET provides flexibility to the assignment until a contractor has to commit to the execution of the task. The same task can be announced several times before its execution, its assignment changing after every announcement.

In our MAS implementation for the dynamic PDPTW, both the vehicle as well as the transportation requests are modeled as agents. In the remainder of this text we will call the agent controlling a vehicle a `VehicleAgent` and the agent responsible for a transportation request an `OrderAgent`. `OrderAgent`s are playing the role of the manager in DynCNET, `VehicleAgent`s are the potential contractors. Figure 11 shows an interaction diagram of an auction using our DynCNET implementation. At the end of an auction, each `VehicleAgent` is either awarded the order or notified

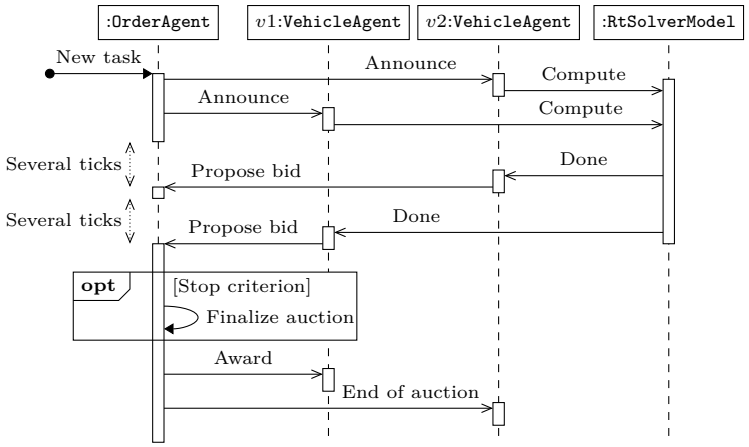


Fig. 11: UML interaction diagram of an auction of an order with two vehicles. Upon receiving the auction announcement, both `VehicleAgent`s start computing a bid. The computations take several ticks. As soon as the `OrderAgent` has met the stop criterion, in this case receiving two bids is enough, the auction is finalized and the order is awarded to $v1$. Vehicle $v2$ is notified of the end of the auction. The `RtSolverModel` lifeline is a simplified view of the implementation, the actual computations can be performed in multiple threads (as discussed in Section 3). Note that the filled arrows indicate synchronous calls and the stick arrows indicate asynchronous calls.

of the end of the auction. At this moment the `VehicleAgent`s have the possibility of starting a new auction by offering one of their previously awarded orders. The `VehicleAgent` will inform the `OrderAgent` responsible for the order that is to be offered to start a new auction, the `OrderAgent` will then perform a new auction process similar to Figure 11. A possible outcome of this auction is that the order is not awarded to another vehicle but stays assigned to the original vehicle. Allowing the vehicles to start a new auction process enables the dynamic (re)allocation of orders and makes our CNET implementation dynamic.

4.2.1 Order agent

The `OrderAgent` (the manager in CNET terminology) is responsible for the auction process. It announces the start of the auction to all vehicles and waits until it receives enough bids to make a decision. The stop criterion for the bidding process is:

$$|bids| \geq 2 \wedge (|bids| = |vehicles| \vee auction\_duration \geq max\_auction\_duration) \quad (12)$$

where, $|bids|$ is the number of received bids, $|vehicles|$ is the total number of vehicles which equals the potential maximum number of bids and $max\_auction\_duration$ is a duration limit that is a parameter of the MAS. An alternative stop criterion would be to wait until all vehicles have submitted their bid, but we use a maximum auction duration because it is more robust in case of (partial) communication loss, vehicle failures, and other unforeseen events.

When the stop criterion evaluates to *true*, the `OrderAgent` finalizes the auction by selecting the best bid as the winner. The best bid is defined as the bid with the lowest price (cost). The order is assigned to the winner, the winner must therefore service that order, unless it decides to auction it and somebody else wins that auction at a later time. All `VehicleAgent`s are informed of the end of the auction. This allows agents that are still computing their bids for this auction to cancel their computations. Bids that are received after the finalization of the auction are ignored.

4.2.2 Vehicle agent

A `VehicleAgent` needs to compute a bid value in order to propose a bid. In our implementation the bids are computed using a solver (the bid solver) managed by the `RtSolverModel`. The cost of an order is defined as the additional cost that including that order incurs to a vehicle's current schedule:

$$cost(order) = cost(new\_schedule) - cost(current\_schedule) \quad (13)$$

where, *current_schedule* is the schedule of the vehicle including all previous order assignments, and *new_schedule* is the current schedule of the vehicle including the proposed order. The task of the solver is finding the best *new_schedule* in a relatively short amount of time to get a reliable estimate of the cost of the auctioned order. The time for computing the new schedule is limited because the auction process has a limited duration, the bid needs to be proposed before the end of this duration in order to ensure that the `OrderAgent` will take the bid into account.

As soon as the assignment of orders to a vehicle has changed, the `VehicleAgent` needs to update its schedule. The vehicle's schedule is optimized by a solver (the schedule solver), although it is imperative to generate a *complete* schedule quickly (e.g. to respond to urgent requests), the solver can compute for a longer time as the solver can continuously notify the `VehicleAgent` of improved schedules. This allows the optimization process to continue for an extended period.

The `VehicleAgent` considers starting a new auction (a reauction) in the following two situations:

- when a vehicle hasn't won an auction for at least five minutes; or,
- when the vehicle's current schedule has changed.

When starting a new auction the vehicle has to decide which of its previously assigned orders it should auction. The order that when removed yields the greatest schedule cost reduction, for that vehicle, is selected. Computing the cost reduction of removing an order from the current route does not require an optimization step (the route is not optimized again) and can therefore be computed quickly for all orders assigned to a vehicle (similar to eq. 13). Orders for which the pickup operation is in process or is already done are not considered for auctioning as they can't be reassigned. If the order with the greatest cost reduction is the last received order, no auction is performed to avoid excessive auctioning. The `VehicleAgent` itself has to propose a bid to its own auction, only when another agent proposes a better bid will the order be reassigned.

4.3 Tuning

For tuning we have run the algorithms on the dataset used by Gendreau et al. [16]. This dataset was chosen because it is very similar to the dataset presented by van Lon and Holvoet [11] as the problem definition is nearly identical. Additionally, using the Gendreau et al. dataset allows comparison with their algorithm. The Gendreau et al. dataset consists of 15 scenarios divided in three different scenario classes (Table 2). It is worth noting that within a scenario class there is quite some variability

Table 2: Characteristics of the three scenario classes of the Gendreau et al. dataset.

Scenario class	Duration	Average request intensity	Fleet size
`4H-24`	4 hours	24 requests per hour	10 vehicles
`4H-33`	4 hours	33 requests per hour	10 vehicles
`7.5H-24`	7.5 hours	24 requests per hour	20 vehicles

of characteristics. For example in `4H-24` the number of orders ranges from 84 to 94, that is an intensity of 21 to 23.5 requests per hour. Similar variability exists in the other scenario classes and also for characteristics such as dynamism and urgency. We performed a benchmark experiment with 28 different OptaPlanner solver configurations. We tested two construction heuristics, first-fit decreasing (FFD) heuristic and cheapest insertion heuristic (CI) and combined each with 14 different local search algorithms. All available local search algorithms provided by OptaPlanner are used with the parameters as advised by the OptaPlanner manual [27]. We used a period of ten seconds for the unimproved time spent stop condition, this is a rather long period that we chose to give OptaPlanner enough time for searching. Since OptaPlanner can be interrupted when the problem changes, there is no downside to this long period. When OptaPlanner is interrupted, it remembers its current best solution, inserts the new problem information, and then continues the search. The algorithms were run three times per scenario, each time with a different random seed. This resulted in a total of $28 \cdot 15 \cdot 3 = 1260$ simulations. Table 3 displays the most relevant results, the complete overview of results can be found in [30]. The best Gendreau et al. algorithms outperform the OptaPlanner algorithms for all scenario classes. This is expected because the Gendreau et al. algorithms are designed and optimized specifically for this problem while the OptaPlanner algorithms are generic

Table 3: Selection of results of the tuning experiment on the Gendreau dataset. Per scenario class, the average cost and rank of a selection of algorithms is shown.

Algorithm	Cost	#
Gendreau	485.6	1
FFD.SHTS	613.8	2
FFD.SA1	614.6	3
FFD.SA4	616.2	4
CI	1023.6	28
FFD	1420.6	29

(a) `4H-24`

Algorithm	Cost	#
Gendreau	3159.4	1
FFD.SHTS	3313.1	2
FFD.LAT	3336.5	3
FFD.SCHC	3337.5	4
CI	5051.7	28
FFD	5778.3	29

(b) `4H-33`

Algorithm	Cost	#
Gendreau	634.8	1
CI.TABU	691.9	2
CI.SA1	693.1	3
FFD.SHTS	702.9	13
CI	1074.4	28
FFD	2656.0	29

(c) `7.5H-24`

Algorithm	Description
Gendreau	The cost values reported are the average of the best algorithm per scenario. The best algorithms proposed by Gendreau et al. [16] are an adaptive descent algorithm and a tabu search algorithm.
CI	Cheapest insertion construction heuristic.
FFD	First-fit decreasing construction heuristic.
FFD.SHTS	FFD with step counting hill climbing with tabu search and strategic oscillation.
FFD.SA1	FFD with simulated annealing with an accepted count limit of 1.
FFD.SA4	FFD with simulated annealing with an accepted count limit of 4.
FFD.LAT	FFD with late acceptance with tabu search.
FFD.SCHC	FFD with step counting hill climbing.
CI.TABU	CI with tabu search.

(d) Algorithm descriptions, details about the algorithms can be found in [27].

local search heuristics. However, the results in Table 3a to 3c indicate that the relative difference between the Gendreau et al. algorithms and the best OptaPlanner algorithm is larger for the small scale scenario (26.4% for `4H-24`) and smaller for the larger scale scenarios (4.9% for `4H-33` and 9% for `7.5H-24`). Since the small scale scenarios in the van Lon and Holvoet [11] dataset are already larger scale than the scenarios in the `4H-24` class, we expect that the performance difference on the van Lon and Holvoet dataset between the Gendreau et al. algorithms and the OptaPlanner algorithms lies between 5 and 9%. This difference is acceptable for the purpose of the current study: analyzing the performance difference between centralized and decentralized usage of the same algorithm. Based on the results we conclude that FFD.SHTS is the best OptaPlanner algorithm to use in the experiments.

4.4 MAS tuning

The MAS has four main parameters that influence the experiment results the most:

- `Bid`, bid solver unimproved time spent.
- `Schedule`, schedule solver unimproved time spent.
- `MAD`, maximum auction duration (eq. 12).
- Reauctions, can be enabled, disabled, or enabled with a cooldown period. The cooldown period is defined as the time that a `VehicleAgent` is not allowed to start a new reauction for an order that was previously reauctioned unsuccessfully.

Choosing the best values for these parameters is especially important because all computational tasks done by agents in the MAS have to be performed on a single

core. This means that in large scale scenarios, 100 agents need to compute their bids on the same core. Using the dataset generator of [11], we created a dataset of large scale scenarios for tuning the MAS. The dataset consists of three levels of dynamism, three levels of urgency, and one level of scale. This gives nine scenario classes, with five scenarios per class, the tuning dataset contains $3 \cdot 3 \cdot 5 = 45$ scenarios. In the first MAS tuning experiment we varied `Bid`. Table 4 shows the MAS settings and the results.

Table 4: First MAS tuning experiment settings and results. Cost is the cumulative cost of the average cost per scenario class, rank is the average rank of the MAS over the nine scenario classes.

`Bid` (ms)	`Schedule` (ms)	`MAD` (s)	Reauctions	Cost	Rank
1	100	5	Enabled	129.1	6.8
2	100	5	Enabled	113.7	5.4
5	100	5	Enabled	105.0	3.9
8	100	5	Enabled	102.2	2.9
10	100	5	Enabled	101.0	2.3
15	100	5	Enabled	102.9	2.2
25	100	5	Enabled	110.8	4.4

The cost and rank values indicate that the `Bid` values 8, 10, and 15 perform best. When looking at the results for these best settings we found that around 66-93% of the auctions are concluded after receiving bids from all agents. Because we expect that receiving less than all bids is not beneficial, we decided to increase the auction duration to ten seconds.

We designed a second experiment that uses the best `Bid` values from last experiment, adds a `Bid` value of 20, tests a higher `Schedule` value, and uses a longer `MAD`. Table 5 shows the settings and the results of the second MAS tuning experiment. The

Table 5: Second MAS tuning experiment settings and results. Cost is the cumulative cost of the average cost per scenario class, rank is the average rank of the MAS over the nine scenario classes.

`Bid` (ms)	`Schedule` (ms)	`MAD` (s)	Reauctions	Cost	Rank
8	100	10	Enabled	102.0	5.4
10	100	10	Enabled	101.3	3.9
15	100	10	Enabled	101.3	3.0
20	100	10	Enabled	104.7	4.4
8	250	10	Enabled	102.5	5.6
10	250	10	Enabled	102.1	4.9
15	250	10	Enabled	103.0	3.7
20	250	10	Enabled	106.2	5.1

results show that a `Schedule` of 100 milliseconds performs better than a `Schedule` of 250 milliseconds. For `Bid` 8, 10, and 15, with `Schedule` 100, the percentage of auctions that are concluded after receiving bids from all agents has increased to 77-97%. It further appears that increasing `MAD` improves performance of `Bid` 15 while performance of `Bid` 8 and 10 is almost unaffected. We decided to keep the `MAD` at 10, and to select the algorithm with the lowest cost and rank, that is a `Bid` of 15 milliseconds and a `Schedule` of 100 milliseconds.

In the third MAS tuning experiment we investigated the effectiveness of reauctions. Table 6 shows the settings and the results of the experiment. As can be seen,

Table 6: Third MAS tuning experiment settings and results. Cost is the cumulative cost of the average cost per scenario class, rank is the average rank of the MAS over the nine scenario classes.

`Bid` (ms)	`Schedule` (ms)	`MAD` (s)	Reauctions	Cost	Rank
15	100	10	Enabled	101.3	3.1
15	100	10	Disabled	103.2	4.1
15	100	10	Cooldown period 1 min.	101.2	2.7
15	100	10	Cooldown period 10 min.	100.8	2.8
15	100	10	Cooldown period 20 min.	100.5	2.3

the performance difference between enabling and disabling reauctions are small. Nevertheless, enabling reauctions performs better compared to disabling them. Using a cooldown period seems to be beneficial, additionally, we found that a longer cooldown period results in a lower number of reauctions and a higher reauction success rate (Table 7). The slightly higher costs when a shorter cooldown period is used appears

Table 7: Third MAS tuning experiment reauction details. Total num. reauctions is the cumulative number of reauctions, success rate is the average reauction success rate per scenario.

Reauctions	Total num. reauctions	Success rate
Enabled	53214	27.2%
Disabled	0	
Cooldown period 1 min.	49533	27.6%
Cooldown period 10 min.	31827	31.8%
Cooldown period 20 min.	26522	32.8%

to be related with the high number of unsuccessful reauctions in this case. We expect that these unsuccessful reauctions and the computations that are involved delay computations for regular auctions, which explains the higher costs in this case. For the main experiments we decided to use a cooldown period of 20 minutes.

5 Evaluation

To evaluate the hypotheses about multi-agent systems and centralized algorithms, our implementations (Section 4) are run using real-time RinSim (Section 3) on the dataset of van Lon and Holvoet [11].

5.1 Experiment setup

The dataset of van Lon and Holvoet [11] has three dimensions: dynamism, urgency, and scale, with three values per dimension that results in a total of 27 different scenario classes. We use ten scenarios for each class and we perform three repetitions per scenario (with different random seeds), this results in a total of $27 \cdot 10 \cdot 3 = 810$

simulations per algorithm. For each dimension there is a hypothesis that addresses the performance of the algorithms on that dimension.

The dataset of van Lon and Holvoet [11] contains scenarios with a length of 8 hours. Since we need to do a large number of experiments in real-time we shortened the scenario length to 4 hours. We used the dataset generator to generate a new dataset and made it available online [30]. The reduction of scenario length was done purely for computational reasons as running such a large number of simulations in real-time takes considerable time. Additionally, the tick size is set to 250 milliseconds and scenarios now require a real-time simulator by default.

Because performance in real-time simulations is hardware dependent, all real-time simulations are performed on the same computer. We used a dedicated Ubuntu machine (version 12.04.5 LTS) with 24 logical cores (two six core Intel Xeon 2.6GHz E5-2630 v2 processors with hyper threading). For the experiments the Java HotSpot 64-Bit Server VM (runtime version: 1.8.0_74-b02) was used. A single simulation requires two logical cores, one for the simulator and one for the solver computations. For the solver computations we used a thread pool of size three, meaning that any additional computations are queued until one of the three threads are available. At least one core needs to be available for the operating system so a maximum number of 11 simulations were run in parallel. Even though no other processes were started on the dedicated computer during the course of the experiments, we opted to use the experiment ordering feature in RinSim (Section 3.4). The factorial setup order that is used is: repetition, scenario, algorithm. This means that the first few simulations are: $r_0 s_0 a_0$, $r_0 s_0 a_1$, $r_0 s_1 a_0$, $r_0 s_1 a_1$, etc. Here r_n stands for random seed (repetition) n, s_n stands for scenario n, and a_n stands for algorithm n. This setup ensures that the execution of the different algorithms is interleaved as much as possible. Additionally, a JVM warm up period of 30 seconds is used to let the JVM perform code optimizations before the actual experiment is started. We choose 30 seconds because this is the default warm up time in OptaPlanner [27]. RinSim nor the algorithms make use of any I/O operations during a simulation. The entire experiment is programmed using Java and is run entirely from memory. Before and after a simulation, RinSim does use I/O operations to read the scenario file and write the results to disk, but since these operations are not done during the simulation itself, this does not influence the results.

For performing the experiments described in this section we used the following software. As simulator, we used RinSim v4.3.0 [31], for the centralized and decentralized algorithms we used RinLog v3.2.2 [32], to generate the scenarios we used the PDPTW dataset generator v1.1.0 [33]. The experiment code including the launch scripts can be found in a separate repository [34]. The datasets that were used in the tuning experiments, the main dataset, as well as all results discussed in this paper, can be found in [30]. We have made sure to archive each of these artifacts, using a Digital Object Identifier (DOI), to ensure their long-term availability.

5.1.1 Algorithms

Based on the tuning experiment (Section 4.3) we selected the FFD.SHTS algorithm that performed best on average on all three scenario classes. Table 8 shows how and with what settings FFD.SHTS is used.

Table 8: OptaPlanner settings used in the centralized and decentralized configurations. The type refers to how the solver is used, the limit is the maximum unimproved time parameter of the OptaPlanner solver.

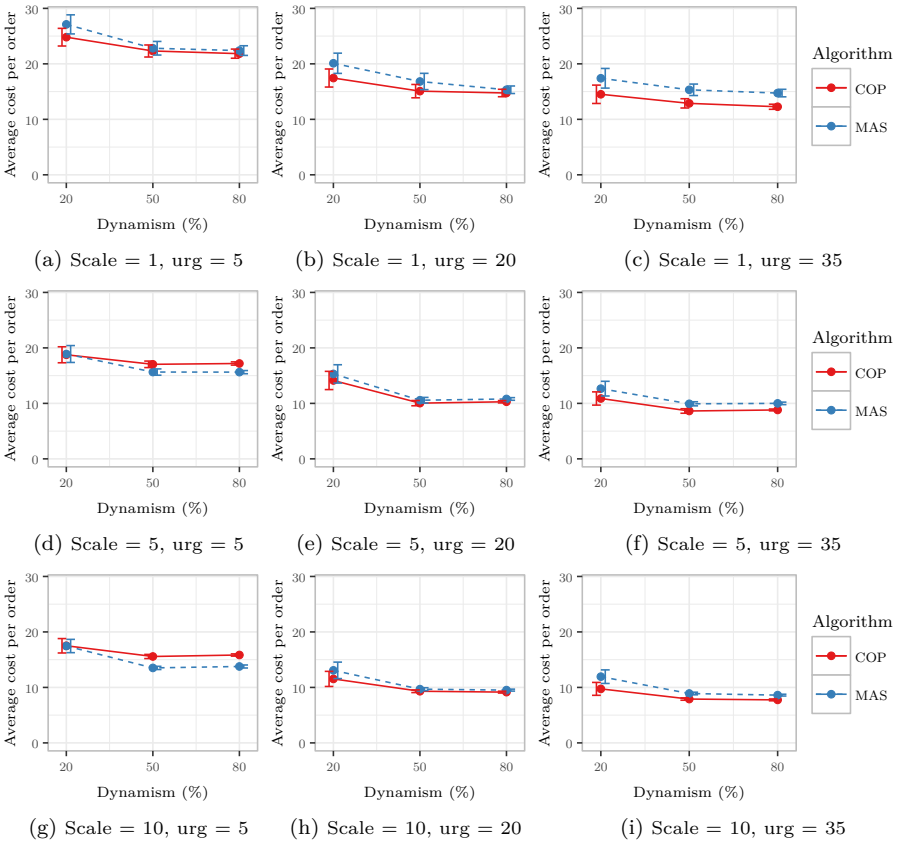
Short name	Name	Type	Limit
MAS	DynCNET	Bid	15 ms
		Schedule	100 ms
COP	Centralized OptaPlanner	Schedule	10000 ms

5.2 Results and analysis

Figures 12-17 display the results of the experiments. Each data point in the graphs is the average of 30 simulations, 10 scenarios from the same class, each repeated 3 times. For each two means obtained under the same settings we analyzed whether they are statistically different using Welch's paired t-test. In the following analysis we refer to this test by mentioning the p-values (when relevant) that were observed. The significance threshold was set at $p = .01$. The cumulative computation time of all 1620 simulations was 1794.4 hours ($\approx$ 74.8 days), since 11 experiments were conducted in parallel, the actual computation time was 165.1 hours ($\approx$ 6.9 days).

5.2.1 Dynamism

Recall the first hypothesis (Section 1): *A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more dynamic problem instances.* More specifically, we can refine this hypothesis into several related sub-hypotheses. First, we expect that more dynamism correlates positively with the average costs for MAS. Figure 12 shows that the costs of MAS remain stable or decrease for every level of scale and urgency when dynamism increases, therefore this hypothesis is rejected. The cost for MAS always decreases significantly between 20% and 50%, between 50% and 80% the decrease is not significant except in Figure 12i ($p \approx 0.006$) and in Figures 12e, 12f, and 12g, where the cost is actually increasing. Second, we expect that more dynamism correlates positively with the average costs for COP. This hypothesis is rejected because the costs of COP are significantly decreasing in Figure 12 between 20% and 50% dynamism (except in Figure 12c), between 50% and 80% the costs difference is never significantly different. These results are similar to the results based on simulated time as reported in [11], this comes as a surprise since we expected that simulating in real-time would make dealing with highly dynamic situations more challenging. However, it turns out that dealing with occasional but relatively big bursts of change is more demanding for the algorithms than more frequent smaller changes. Third, we expect that increasing dynamism increases the cost of MAS less than that it increases the cost of COP. Figure 13 shows that in many situations, MAS performs relatively better when dynamism increases. In Figures 13c, 13e, and 13f, however, this is not the case. In very urgent and large scale scenarios, the hypothesis can be accepted because the performance of MAS decreases more relative to COP. This result is contrary to the experiments performed by van Lon and Holvoet [11] and van Lon et al. [19] where dynamism had almost no influence. We attribute this difference in results to the fact that in the present paper the experiments were conducted using real-time simulation, contrary to the simulations in [11, 19]. This is confirmed by the fact that this effect only occurs in



(a) Scale = 1, urg = 5 (b) Scale = 1, urg = 20 (c) Scale = 1, urg = 35

(d) Scale = 5, urg = 5 (e) Scale = 5, urg = 20 (f) Scale = 5, urg = 35

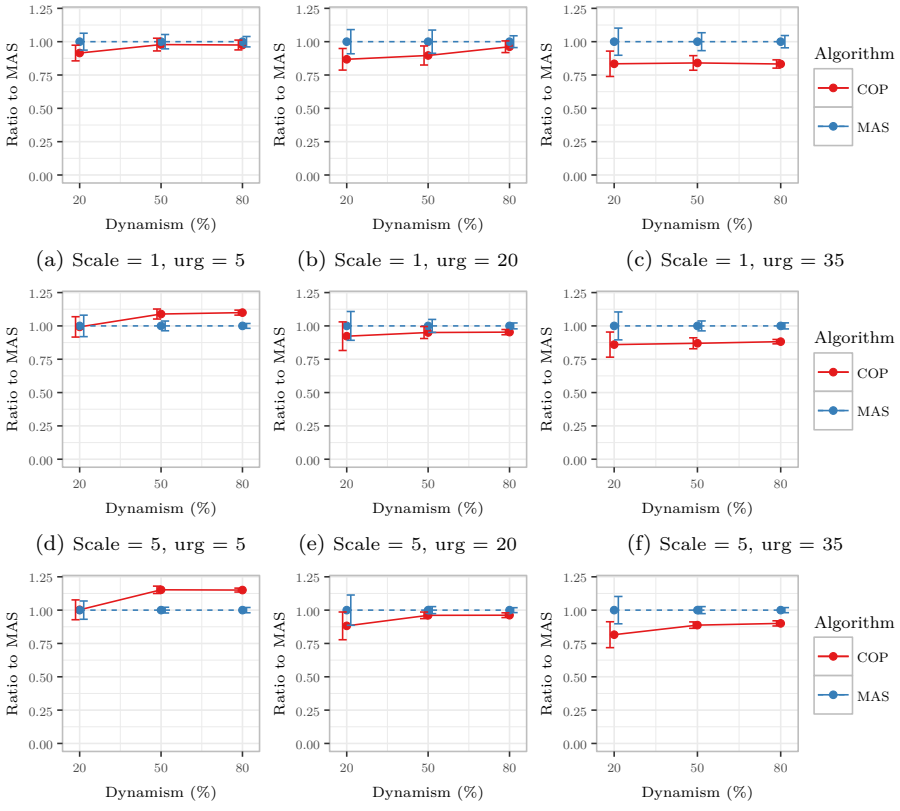
(g) Scale = 10, urg = 5 (h) Scale = 10, urg = 20 (i) Scale = 10, urg = 35

Fig. 12: Comparison with mean relative cost versus dynamism for all levels of scale and urgency. The error bars indicate a confidence interval of 99%.

very urgent and in large scale situations, as these are the conditions where real-time simulation has the biggest impact on the available computation time. In general, we can conclude, based on this experiment, that dynamism influences the *relative* performance of COP and MAS in very urgent and in large scale scenarios. In absolute terms, however, lower dynamism tends to induce higher average costs.

5.2.2 Urgency

The second hypothesis (Section 1) concerns urgency: *A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on more urgent problem instances*. Regarding urgency, we expect that more urgent problems (lower urgency values) are correlated with higher average costs per order for MAS and COP. The results (Figure 14) show that for both COP and MAS this is true. Between 5 and 20 minutes the difference in cost is always significant. Between 20 and 35 minutes, this is not the case for COP in Figures 14a and 14g, and for MAS, the differences are not significant between 20 and 35 minutes in Figures 14a, 14b, 14c, 14d, and 14g. These results and the figures show a diminishing of the cost decreases



(a) Scale = 1, urg = 5 (b) Scale = 1, urg = 20 (c) Scale = 1, urg = 35

(d) Scale = 5, urg = 5 (e) Scale = 5, urg = 20 (f) Scale = 5, urg = 35

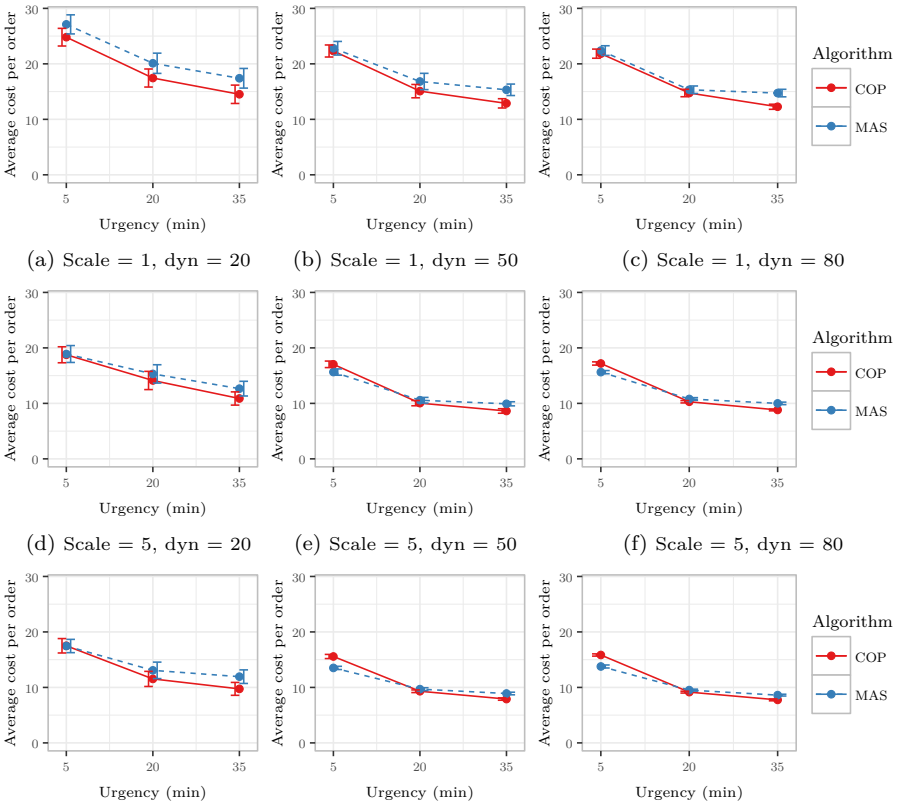
(g) Scale = 10, urg = 5 (h) Scale = 10, urg = 20 (i) Scale = 10, urg = 35

Fig. 13: Comparison with competitive ratio to MAS versus dynamism for all levels of scale and urgency. The error bars indicate a confidence interval of 99%.

the less urgent a problem gets. However, it appears that this effect is stronger for MAS than for COP. The relative cost of MAS versus COP (Figure 15) shows that less urgency (higher values) benefits COP more than it benefits MAS. Similarly, COP suffers to a greater extent from more urgency than MAS. We have to reject the hypothesis, however, because there are only four very urgent cases where MAS outperforms COP significantly (Figures 15e, 15f, 15h, and 15i). In four other very urgent situations, the costs of MAS and COP are not significantly different (Figures 15b, 15c, 15d, and 15g). Therefore, MAS is not always better at responding to the most urgent requests. More generally, it seems that MAS is better at coping with a simultaneous increase of urgency, dynamism, and scale compared to COP. This indicates that MAS is better at coping with a continuously changing and large scale problem that requires quick decisions.

5.2.3 Scale

The third hypothesis: *A CNET based MAS finds solutions with a lower operating cost compared to a centralized algorithm on larger scale problem instances*, raises several



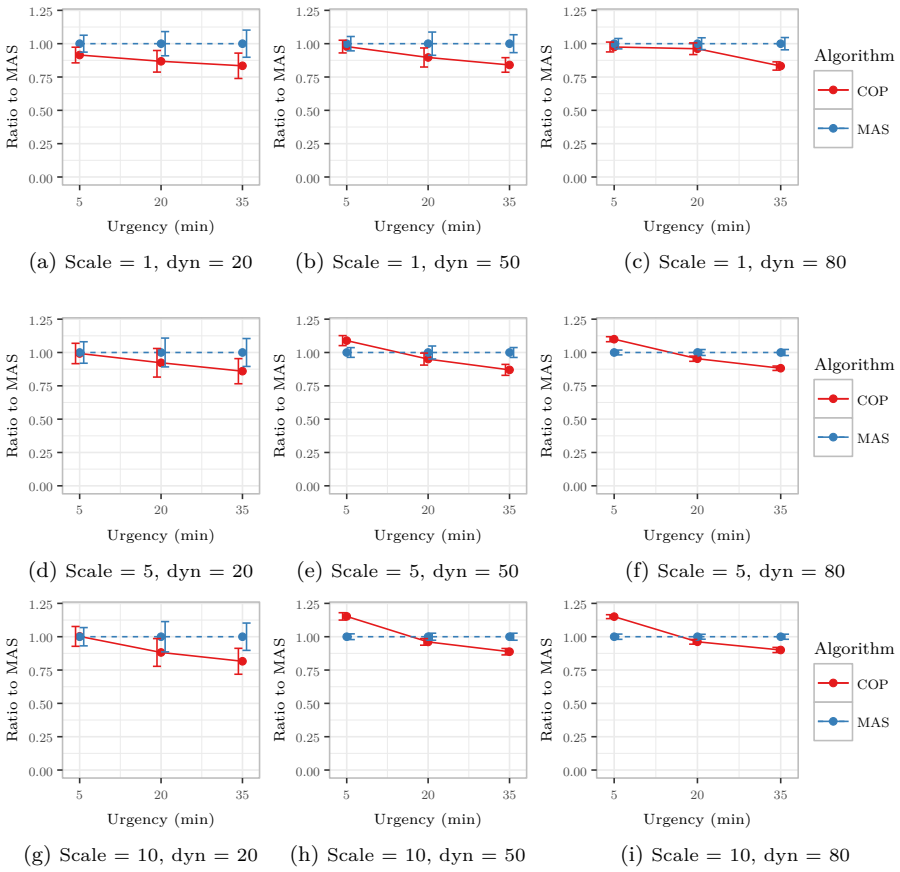
(a) Scale = 1, dyn = 20 (b) Scale = 1, dyn = 50 (c) Scale = 1, dyn = 80

(d) Scale = 5, dyn = 20 (e) Scale = 5, dyn = 50 (f) Scale = 5, dyn = 80

(g) Scale = 10, dyn = 20 (h) Scale = 10, dyn = 50 (i) Scale = 10, dyn = 80

Fig. 14: Comparison with mean relative cost versus urgency for all levels of scale and dynamism. The error bars indicate a confidence interval of 99%.

related hypotheses. First, when scaling up a problem it is expected that, because of the larger solution space, algorithms have more difficulty of finding good solutions. Therefore, we expect that the average cost per order increases for larger scale problems. Figure 16 shows that this is not true for MAS. The average cost for scale 5 is always significantly lower than the average cost for scale 1, this is similarly the case between scale 5 and scale 10, except for Figures 16a and 16g where the difference is not significant ($p \approx .027$ and $p \approx .348$ respectively). For COP the situation is similar, there are only two cases where the cost is not decreasing significantly, between scale 5 and 10 in Figure 16a ($p \approx .050$) and in Figure 16g ($p \approx .102$). This leads us to reject the hypothesis for both COP and MAS. Although seemingly counter intuitive, these results are logical when considering the fact that with larger scale problems both the number of vehicles and the number of orders increase. Since there are more vehicles, the average distance of a new arriving order to the closest vehicle will be lower. This has a positive effect on the tardiness and distance traveled. The results indicate that both algorithms have enough time to explore the search space to exploit the larger number of available vehicles. Figure 17 shows that the relative performance of COP and MAS depends on the level of dynamism and urgency. For



(a) Scale = 1, dyn = 20 (b) Scale = 1, dyn = 50 (c) Scale = 1, dyn = 80

(d) Scale = 5, dyn = 20 (e) Scale = 5, dyn = 50 (f) Scale = 5, dyn = 80

(g) Scale = 10, dyn = 20 (h) Scale = 10, dyn = 50 (i) Scale = 10, dyn = 80

Fig. 15: Comparison with competitive ratio to MAS versus urgency for all levels of scale and dynamism. The error bars indicate a confidence interval of 99%.

example, in very urgent and medium to very dynamic situations (Figures 17b and 17c), MAS benefits more from an increasing scale compared to COP, while in not so dynamic situations, this trend seems to be reverse (Figures 17d and 17g). In Figure 17f the trend appears approximately parallel. Based on these differing trends we cannot accept the hypothesis that MAS is generally more scalable than COP.

5.3 Discussion

All three hypotheses, about MASs being better to cope with increasing dynamism, urgency, and scale, compared to centralized algorithms have been rejected. However, the reverse hypotheses can neither be accepted. The results are more nuanced, the centralized algorithm is better in most situations but there exist specific problems that are very dynamic, very urgent, and large scale, for which MASs are better. Table 9 shows that of the 27 different settings in the experiment, there are four settings where MAS significantly outperforms COP, 18 settings where COP significantly outperforms MAS, and five settings where the difference is not significant. The four

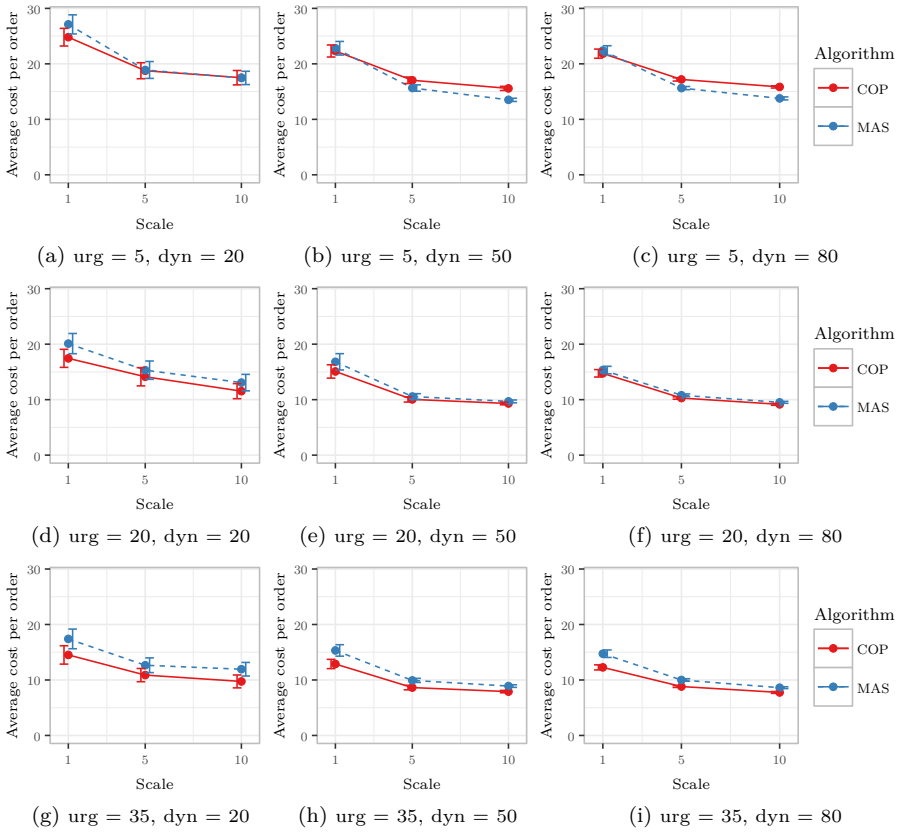
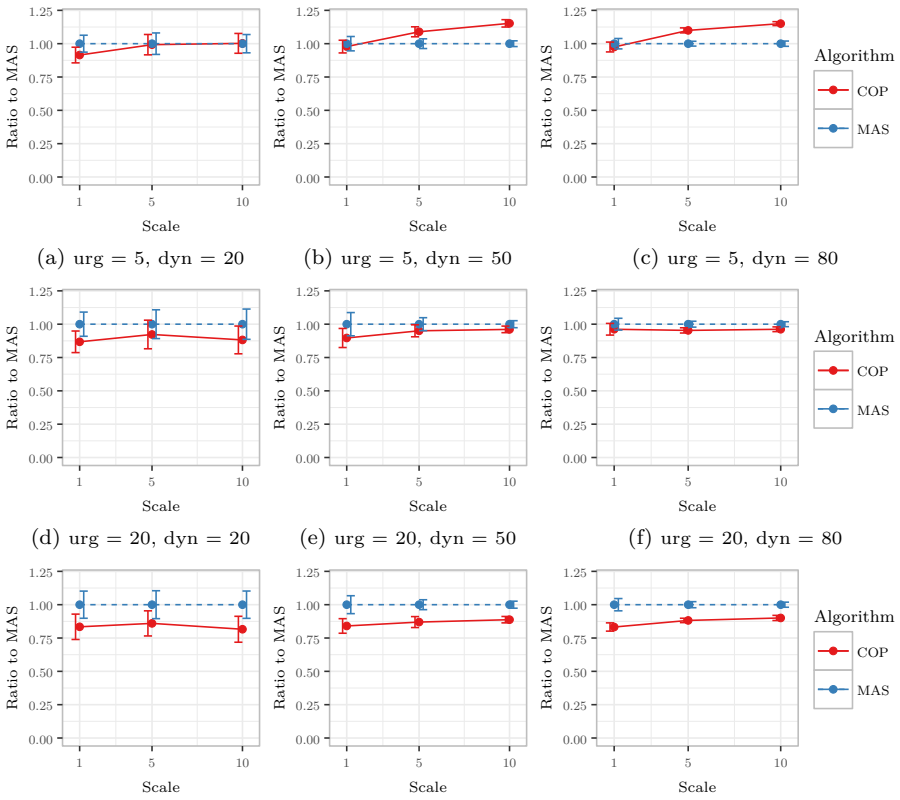


Fig. 16: Comparison with mean relative cost versus scale for all levels of urgency and dynamism. The error bars indicate a confidence interval of 99%.

settings where MAS outperforms COP all have an urgency of 5 minutes. This indicates that the advantage of COP of having a global view on the problem diminishes for very urgent problems. For very urgent problems, COP may not have enough time for searching the solution space, apparently, the implicit solution space partitioning of the CNET algorithm helps finding a good solution in a short amount of time. This is interesting because centralized algorithms can also benefit from this knowledge, for example, it would be interesting to experiment with a similar partitioning but in a centralized setting.

The average COP-to-MAS ratio is 0.942, meaning that when COP operates the fleet of vehicles it costs, on average, only 94.2% relative to MAS. This means that in general, COP is the preferred solution approach. However, MAS is a better fit if it is known beforehand that the problem is very urgent (ratio 1.039), very urgent and medium to large scale (ratio 1.081), or, medium to very dynamic, very urgent, and medium to large scale (ratio 1.123).

In practice, the deployment of the algorithms under investigation is relevant. A benefit of MASs is their ability to be deployed decentrally as well as centrally, this allows MASs to replace algorithms in an existing centralized deployment. Addi-



(a) urg = 5, dyn = 20 (b) urg = 5, dyn = 50 (c) urg = 5, dyn = 80

(d) urg = 20, dyn = 20 (e) urg = 20, dyn = 50 (f) urg = 20, dyn = 80

(g) urg = 35, dyn = 20 (h) urg = 35, dyn = 50 (i) urg = 35, dyn = 80

Fig. 17: Comparison with competitive ratio to MAS versus scale for all levels of urgency and dynamism. The error bars indicate a confidence interval of 99%.

tionally, due to their decentralized nature, parallelizing MASs is trivial, in fact the current implementation is already multi-threaded (although in the experiments limited to a single thread) but executed on a single core. Although out of scope of the current study, running the MASs using multiple cores would theoretically decrease the average costs per order. Additionally, MASs can be deployed in a distributed setting using smartphones to run the agents on. This allows a purely decentralized setup that is robust to hardware failure. If some hardware fails (e.g. a smartphone) it would only affect one vehicle and the orders that it has won in an auction. These effects could even be reduced by implementing a protocol between a vehicle agent and order agent that frequently checks whether it is still alive, similarly to pheromone evaporation in Delegate MAS [35].

6 Conclusion

A widely held belief in multi-agent systems literature is that MAS is advantageous in operational research problems that are very dynamic and/or large scale. However, such claims were never supported by evidence based on a systematic empirical study.

Table 9: Average results of both COP and MAS for each setting. The 'Best' column indicates the best performing algorithm for that scenario class, a † indicates that the difference is not statistically significant ($p < 0.01$).

Dynamism	Urgency	Scale	COP	MAS	Ratio	p-value	Best
20	5	1	24.804	27.115	0.915	1.683e-07	COP
50	5	1	22.318	22.813	0.978	7.068e-02	COP†
80	5	1	21.836	22.392	0.975	7.209e-02	COP†
20	20	1	17.444	20.101	0.868	7.191e-11	COP
50	20	1	15.080	16.821	0.896	3.908e-07	COP
80	20	1	14.747	15.330	0.962	1.913e-02	COP†
20	35	1	14.514	17.399	0.834	9.080e-12	COP
50	35	1	12.878	15.322	0.840	1.390e-11	COP
80	35	1	12.267	14.733	0.833	2.128e-10	COP
20	5	5	18.758	18.896	0.993	1.227e-01	COP†
50	5	5	17.038	15.644	1.089	2.690e-14	MAS
80	5	5	17.188	15.629	1.100	2.293e-16	MAS
20	20	5	14.128	15.310	0.923	3.487e-13	COP
50	20	5	10.039	10.562	0.950	4.042e-07	COP
80	20	5	10.290	10.794	0.953	2.509e-07	COP
20	35	5	10.878	12.652	0.860	1.770e-15	COP
50	35	5	8.628	9.920	0.870	7.927e-16	COP
80	35	5	8.817	9.997	0.882	1.809e-12	COP
20	5	10	17.506	17.463	1.002	6.332e-01	MAS†
50	5	10	15.574	13.513	1.153	8.956e-25	MAS
80	5	10	15.843	13.771	1.150	1.886e-21	MAS
20	20	10	11.531	13.074	0.882	1.099e-14	COP
50	20	10	9.308	9.685	0.961	1.686e-06	COP
80	20	10	9.150	9.511	0.962	3.054e-06	COP
20	35	10	9.736	11.936	0.816	5.141e-20	COP
50	35	10	7.901	8.901	0.888	2.529e-18	COP
80	35	10	7.752	8.609	0.900	1.268e-14	COP

Present paper is the first to systematically investigate the influence of dynamism, urgency, and scale on the performance of both multi-agent systems and centralized algorithms. Based on the experimental results, we reject the hypotheses that the MAS has a lower average operating cost compared to the centralized algorithm on more dynamic, more urgent, or larger scale problems. However, the reverse hypotheses can neither be accepted. The results are more nuanced, the solutions found by the centralized algorithm cost, on average, only 94.2% of the cost of the solutions found by the multi-agent system. This indicates that the centralized algorithm generally performs better compared to the multi-agent system. However, for scenarios that are medium to very dynamic, very urgent, and medium to large scale, the average relative cost of the centralized algorithm is 112.3%, indicating that under these circumstances, the multi-agent system performs better than the centralized algorithm. When assessing the performance of the algorithms individually per scenario property, there is not one algorithm that generally outperforms the other on that dimension.

Running an empirical study for comparing distinct algorithms is a tedious task. We have formally defined the pickup-and-delivery problem, including the scenario properties: dynamism, urgency, and scale. For the algorithms we used OptaPlanner, a well known satisfaction solver library. A tuning experiment was conducted to find the best performing OptaPlanner algorithm for this problem. The best algorithm was incorporated in an online centralized algorithm and a multi-agent system based on the dynamic contract-net protocol. In order to perform a fair empirical study it

is imperative to use a real-time simulator that assigns the same processing power to the approaches. For this reason we have extended the RinSim logistics simulator and have demonstrated that fluctuations caused by the real-time nature of the simulator have a minimal impact on the end result.

To facilitate complete reproducibility, the simulator, datasets, algorithms, and results are available online. This allows some interesting directions for future work. For example, the algorithms could be evaluated on different problem instances generated with our dataset generator or on different problems in the field of logistics. Additionally, there are many other MAS coordination protocols such as Delegate MAS or Gradient Field, that can be evaluated and compared to the algorithms used in the present paper. Similarly, there are many more centralized algorithms and libraries that implement them. Present paper provides a benchmark, an ideal starting point for further research into more advanced algorithms. During the realization of this article the authors published an investigation into optimizing multi-agent systems with genetic programming [36] and evaluated it using the benchmark presented in this paper.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven.

References

1. M. Wooldridge. *An introduction to multiagent systems*. Wiley, 2002.
2. Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.
3. Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17(3):397–431, 2008. ISSN 1573-7454. doi:10.1007/s10458-008-9050-0.
4. Danny Weyns, Kurt Schelfthout, Tom Holvoet, and Tom Lefever. Decentralized control of e'gv transportation systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 67–74, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi:10.1145/1082473.1082806.
5. Klaus Dorer and Monique Calisti. An adaptive solution to dynamic transport optimization. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 45–51, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi:10.1145/1082473.1082803.
6. Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202 (1):8–15, 2010. ISSN 03772217. doi:10.1016/j.ejor.2009.04.024.
7. Klaus Fischer, Jörg P. Müller, and Markus Pischel. A model for cooperative transportation scheduling. In *Proc. of the 1st Int. Conf. on Multiagent Systems (ICMAS'95)*, pages 109–116, San Francisco, 1995.

8. Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980. ISSN 0018-9340. doi:10.1109/TC.1980.1675516.
9. Martijn Mes, Matthieu van der Heijden, and Aart van Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1):59–75, 2007. ISSN 0377-2217. doi:10.1016/j.ejor.2006.02.051.
10. Tamás Máhr, Jordan F. Srour, Mathijs de Weerdt, and Rob Zuidwijk. Agent performance in vehicle routing when the only thing certain is uncertainty. In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, Estorial, Portugal, 2008.
11. Rinde R. S. van Lon and Tom Holvoet. Towards systematic evaluation of multi-agent systems in large scale and dynamic logistics. In Qingliang Chen, Paolo Torroni, Serena Villata, Jane Hsu, and Andrea Omicini, editors, *PRIMA 2015: Principles and Practice of Multi-Agent Systems: 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*, pages 248–264. Springer International Publishing, Cham, 2015. ISBN 978-3-319-25524-8. doi:10.1007/978-3-319-25524-8_16.
12. Támas Máhr, Jordan F. Srour, Mathijs de Weerdt, and Rob Zuidwijk. Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research: Part C*, 18(1):99–119, 2010. doi:10.1016/j.trc.2009.04.018.
13. Darrel C. Ince, Leslie Hatton, and John Graham-Cumming. The case for open computer programs. *Nature*, 482(7386):485–488, 2012. ISSN 1476-4687. doi:10.1038/nature10836.
14. Rinde R. S. van Lon and Tom Holvoet. Evolved multi-agent systems and thorough evaluation are necessary for scalable logistics. In *2013 IEEE Workshop on Computational Intelligence In Production And Logistics Systems (CIPLS)*, pages 48–53. 2013. doi:10.1109/CIPLS.2013.6595199.
15. Victor Pillac, Michel Gendreau, Christelle Gueret, and Andres L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013. ISSN 0377-2217. doi:10.1016/j.ejor.2012.08.015.
16. Michel Gendreau, François Guertin, Jean-Yves Potvin, and René Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14 (3):157–174, 2006. ISSN 0968090X. doi:10.1016/j.trc.2006.03.002.
17. O. B G Madsen, Hans F. Ravn, and Jens Moberg Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995. ISSN 02545330. doi:10.1007/BF02031946.
18. J Yang, P Jaillet, and H Mahmassani. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2):135–148, 2004. ISSN 0041-1655. doi:10.1287/trsc.1030.0068.
19. Rinde R. S. van Lon, Eliseo Ferrante, Ali E. Turgut, Tom Wenseleers, Greet Vanden Berghe, and Tom Holvoet. Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, 253(3):614–624, 2016. ISSN 0377-2217. doi:10.1016/j.ejor.2016.03.021.
20. Rinde R. S. van Lon and Tom Holvoet. RinSim: A simulator for collective adaptive systems in transportation and logistics. In *Proceedings of the 6th IEEE*

International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012), pages 231–232, Lyon, France, 2012. doi:10.1109/SASO.2012.41.
21. Averill M. Law. *Simulation modeling and analysis*. McGraw-Hill, fourth edition, 2007.
22. Thomas Preisler, Tim Dethlefs, and Wolfgang Renz. Data-adaptive simulation: Cooperativeness of users in bike-sharing systems. In Wolfgang Kersten, Thorsten Blecker, and Christian M. Ringle, editors, *Innovations and Strategies for Logistics and Supply Chains*, pages 1765–1772, 2015.
23. Thomas Preisler, Tim Dethlefs, and Wolfgang Renz. Self-organizing redistribution of bicycles in a bike-sharing system based on decentralized control. In *Federated Conference on Computer Science and Information Systems*, volume 8, pages 1471–1480. ACSIS, 2016. doi:10.15439/2016F126.
24. Jonathan Merlevede, Rinde R.S. van Lon, and Tom Holvoet. Neuroevolution of a multi-agent system for the dynamic pickup and delivery problem. In *International Joint Workshop on Optimisation in Multi-Agent Systems and Distributed Constraint Reasoning (co-located with AAMAS)*, 2014.
25. Rinde R. S. van Lon, Tom Holvoet, Greet Vanden Berghe, Tom Wenseleers, and Juergen Branke. Evolutionary Synthesis of Multi-Agent Systems for Dynamic Dial-a-Ride Problems. In *GECCO Companion '12 Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 331–336, Philadelphia, USA, 2012. ISBN 9781450311786. doi:10.1145/2330784.2330832.
26. Hoang Tung Dinh, Rinde R. S. van Lon, and Tom Holvoet. Multi-Agent Route Planning Using Delegate MAS. In *ICAPS Proceedings of the 4th Workshop on Distributed and Multi-Agent Planning (DMAP-2016)*, pages 24–32, 2016.
27. Geoffrey De Smet et al. *OptaPlanner User Guide*. Red Hat and the community. URL `http://www.optaplanner.org`. OptaPlanner is an open source constraint satisfaction solver in Java.
28. Weiming Shen, Qi Hao, Hyun Joong Yoon, and Douglas H. Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4):415 – 431, 2006. ISSN 1474-0346. doi:10.1016/j.aei.2006.05.004.
29. Danny Weyns, Nelis Boucké, Tom Holvoet, and Bart Demarsin. DynCNET: A protocol for dynamic task assignment in multiagent systems. *First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007*, pages 281–284, 2007. doi:10.1109/SASO.2007.20.
30. Rinde R.S. van Lon. When do agents outperform centralized algorithms? - A systematic empirical evaluation in logistics - datasets and results v1.1.0, May 2017. doi:10.5281/zenodo.576345.
31. Rinde R.S. van Lon. RinSim v4.3.0, December 2016. URL `https://github.com/rinde/RinSim/tree/v4.3.0`. doi:10.5281/zenodo.192106.
32. Rinde R.S. van Lon. RinLog v3.2.2, May 2017. URL `https://github.com/rinde/RinLog/tree/v3.2.2`. doi:10.5281/zenodo.571180.
33. Rinde R.S. van Lon. PDPTW dataset dataset: v1.1.0, August 2016. URL `https://github.com/rinde/pdptw-dataset-generator/tree/v1.1.0`. doi:10.5281/zenodo.59259.
34. Rinde R.S. van Lon. When do agents outperform centralized algorithms? - A systematic empirical evaluation in logistics - code v1.1.0, May 2017. URL `https://github.com/rinde/vanLon17-JAAMAS-code`. doi:10.5281/zenodo.576389.

35. Tom Holvoet, Danny Weyns, and Paul Valckenaers. Patterns of delegate mas. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 1–9, Sept 2009. doi:10.1109/SASO.2009.31.
36. Rinde R. S. van Lon, Juergen Branke, and Tom Holvoet. Optimizing agents with genetic programming: an evaluation of hyper-heuristics in dynamic real-time logistics. *Genetic Programming and Evolvable Machines*, pages 1–28, 2017. ISSN 1573-7632. doi:10.1007/s10710-017-9300-5.